ARTIFICIAL INTELLIGENCE

(CSC 462)

ASSIGNMENT # 1

NAME: MUAAZ BIN MUKHTAR

REG NO: FA21-BSE-045

CLASS & SECTION: BSSE-5A

SUBMITTED TO: SIR WAQAS ALI

DATE SUBMITTED: 03-10-2023

Department of Computer Science

QUESTION 1

Problem Statement: A person needs to drive to the city airport early in the morning without prior knowledge of route assistance from anyone.

Operators:

Primary operator is *driving a car*.

Additional operators include *choosing the path* or *making decisions at junctions*.

Solution Space: It consists of all the possible routes a person can use to reach the destination i.e. Airport. Each decision point (road junction) adds branches to solution space, making it very large.

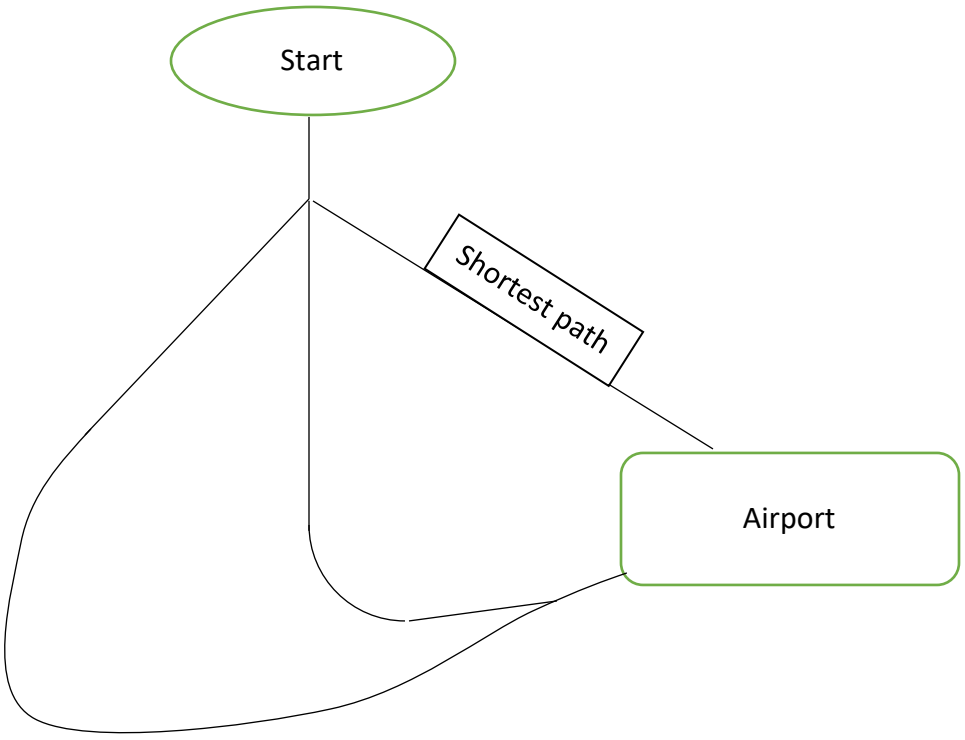
Goal State: Arriving at the city Airport.

Search Strategy: It is not a good decision to follow a blind search strategy in this scenario. In this scenario the solution set will be very large, so blind strategy will be very inefficient and may result in getting lost or may take a lot of time.

However, heuristic search strategy would be appropriate, which may include

- Following highways
- Following signboards
- Asking directions from locals or markets open at that time

Graphical Representation:



QUESTION 2:

WSP	ISP
• Clear and well-defined statement. • Mostly have single solution. • Problem Solving steps are straightforward and can be specified in advance. • Amendable to systematic and algorithmic approaches. • Solutions are based on established formulas. • E.g. solving equations 2+2=4, shortest path etc.	• Lack clear and well-defined statement. • Open-ended, ambiguous and complex. • Multiple valid interpretations and correct solutions may exist. • Require creative and heuristic approaches. • Involves judgement, creativity etc. • E.g. forming semester schedule, resolving social issues etc.

QUESTION 3:

Applying BFS:

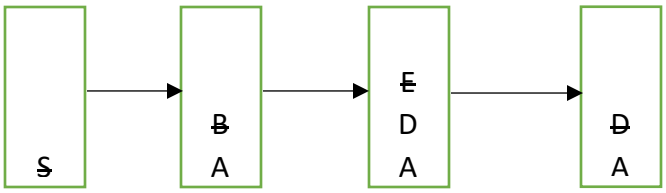
FIFO queue: ~~S, A, B, C, D~~, E, F

Visited: S A B C D

Path: S, A, B, C, D

Applying DFS:

LIFO stack:



Visited: S B D F

Path: S, B, E, back trace to B, D

QUESTION 4:

DFS advantage: Memory efficient as it explores deeply before back tracing.

DFS disadvantage: Can stuck in deep branches.

BFS advantage: Guarantees shortest path.

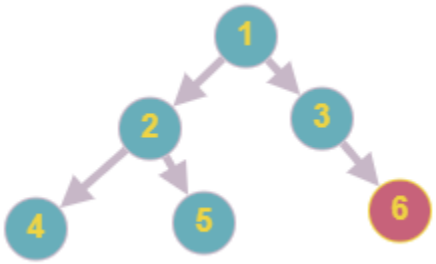
BFS disadvantage: Can be memory intensive.

Progressive deepening approach:

- Starts from depth limit of 1 (BFS) and performs DFS.
- If solution is not found, increase the limit by 1 and perform DFS.

- Continue this process till solution is found.

Example:



First it will explore D, then d2 and then d3 to find solution.

QUESTION 5:

Problems in Hill Climbing:

Local Maxima:

Problem: Getting stuck in suboptimal solutions.

Solution: Random restart, Simulated annealing, Genetic algos.

Plateau Problem:

Problem: Slow progress on flat regions.

Solution: Simulated annealing, Tabs search, Mutation.

Ridge Problem:

Problem: Difficulty navigating narrow steep regions.

Solution: PCA transformation, Partial Restart, Heuristic guidance.

Solution to given tree:

Assuming that the algorithm starts at the root node, the following steps will be taken:

- The algorithm will choose the child node with the best estimate of remaining distance to the goal, which is the node with the value 9.
- The algorithm will then choose the child node of the node with the value 9 with the best estimate of remaining distance to the goal, which is the node with the value 7.
- The algorithm will then choose the child node of the node with the value 7 with the best estimate of remaining distance to the goal, which is the node with the value 5.
- The algorithm will then choose the child node of the node with the value 5 with the best estimate of remaining distance to the goal, which is the goal state.
- The algorithm will now terminate because it has reached the goal state.

Handling the common problems in the given example

Local maximum: If the algorithm gets stuck in a local maximum, it can use the random restart or backtracking solutions.

Plateau: The given tree does not contain any plateaus, so this problem is not relevant in this case.

Ridge: The given tree does not contain any ridges, so this problem is not relevant in this case.

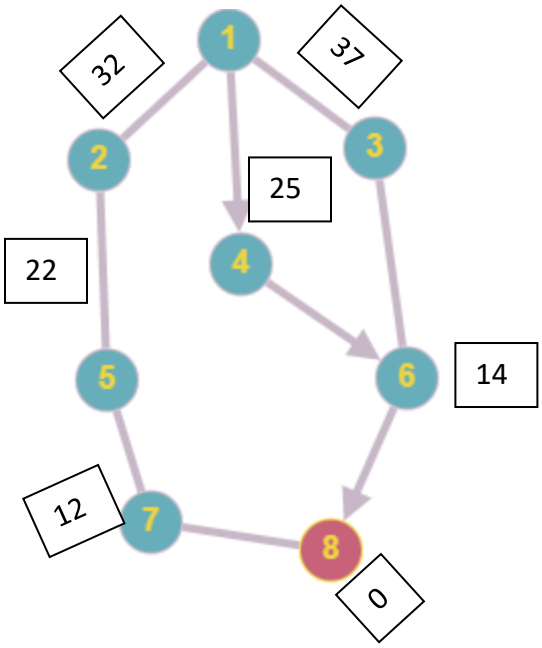
QUESTION 6:

Best First Search (BFS): It is an informed search algorithm that explores a tree or a graph by selecting the most promising node based on some heuristic value.

Working:

- Initialization: Start at initial node (root) and evaluate using heuristic approach to estimate its proximity to goal.
- Expansion: Select node with best heuristic value from open list, this node becomes current node.
- Goal Test: Check if current node is goal.
- Successor Generation: Generate childs of current node and calculate heuristic value.
- Adding to open: Add child to open list based on heuristic value.
- Repeat: Repeat steps 2-5, until goal is found or list becomes empty.

Example:



Open List:

~~A C~~ B D

~~F~~ E B D

~~G~~ E B D

Path: A, C, F, G

- BFS is not always best strategy, its success heavily depends on quality of heuristic formula.
- If heuristic function is admissible and consistent, then it will give optimal solution.

QUESTION 7:

Open List:

~~S A~~ B

~~C~~ D B

~~H~~ G D B

~~L~~ K D B

Path: S, A, C, H, L
Optimal Path: S, A, C, H, L

Beam Search with Beam value is equal to 3 is not equal to BFS with B=1, because here a max of 3 nodes is kept in memory unlike the B=1.

QUESTION 8:

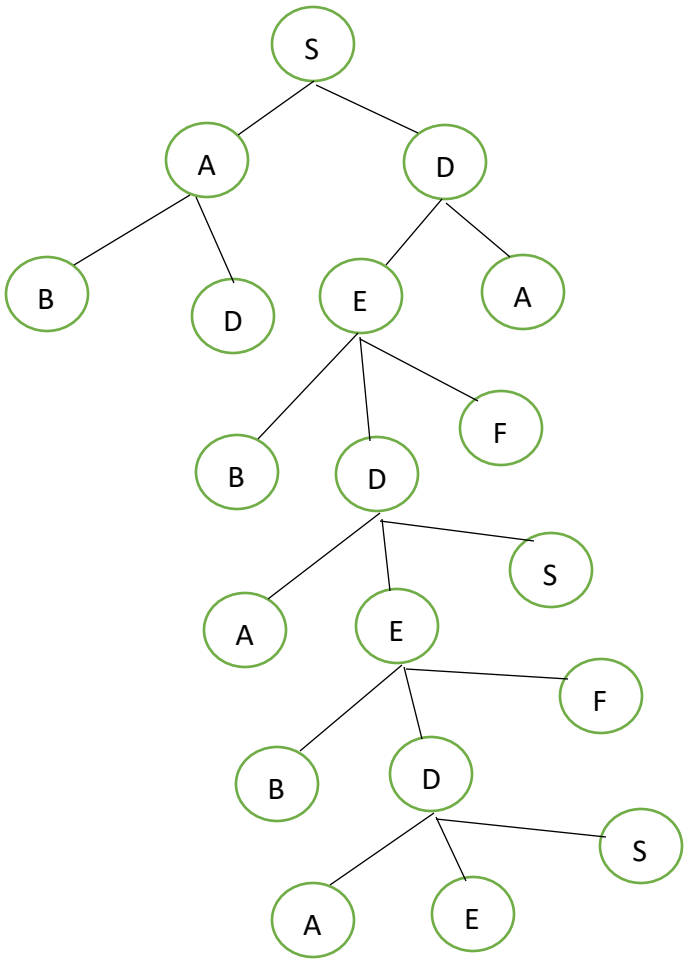
Branch and Bound Strategy:

- **Divide:** Problem is divided into smaller sub-problems represented in tree structure. Each node in tree represents sub-problem that needs to be solved.
- **Bound:** For each sub-problem, a lower bound on optimal solution value is calculated. The lower bound serves as threshold value for deciding whether to explore that branch or not. If lower bound of sub-problem is worse than best solution found so far, that branch is pruned, as it can't lead to optimal solution.
- **Search:** The algorithm explores the tree of sub-problems using any search strategy particularly BFS and DFS.
- **Update:** As the search progresses, the best solution found is updated, whenever a best solution is encountered.

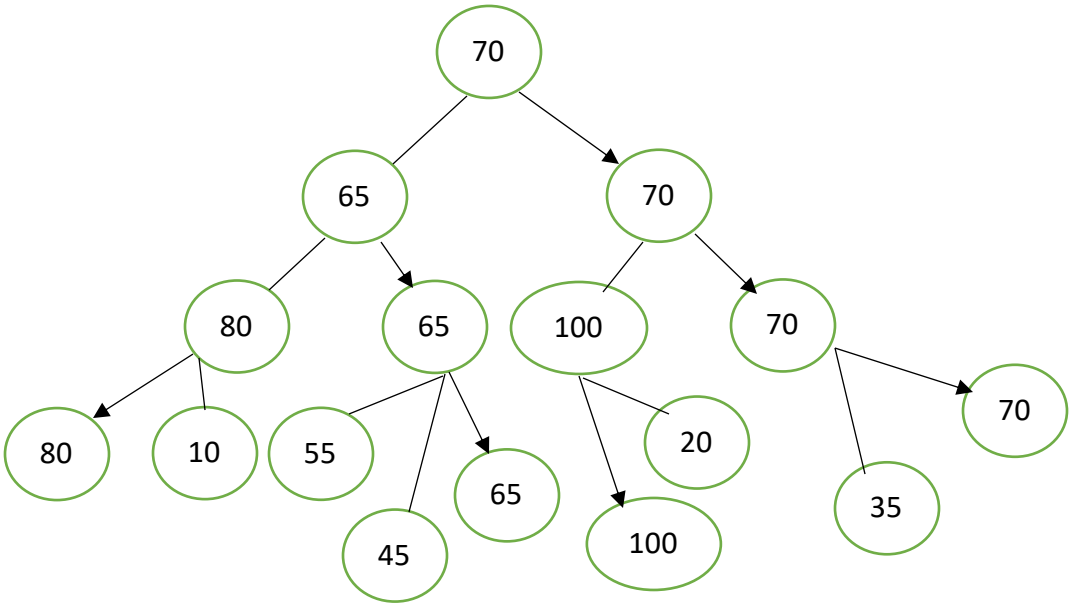
Improvements in Branch and Bound Algorithm:

- Enhancing accuracy of bounding techniques using linear programing, relaxations or heuristics.
- Improving branching strategies by seeding sub-problems that are more optimal likely to lead to solution.
- Utilizes parallel processing to explore different Utilize branches simultaneously, increasing algos speed & reducing time complexity.
- Implementing memory efficient data structures.

Solving the Graph:



QUESTION 9:



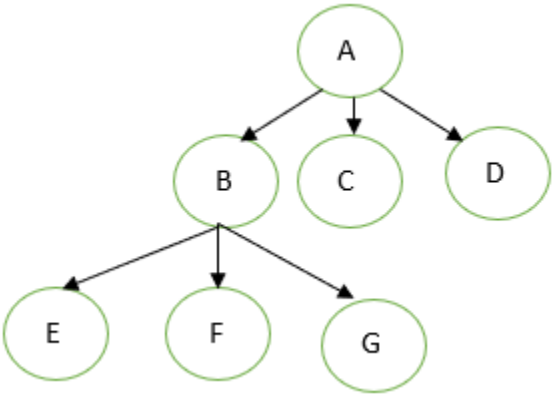
First Layer: Max
Second Layer: Min
Third Layer: Max

QUESTION 10:

Alpha-Beta Pruning: It is a technique used in game trees and search algorithms to reduce no of static evaluations at leaf nodes by efficiently pruning branches that are guaranteed not to have affect final result.
Alpha: The best value found so far for maximizing player.
Beta: The best value found so far for minimizing player.
It exploits the following features:

- **Initial Values:** At root level, alpha is set to negative infinity, and beta is set to positive infinity. This ensures that any value encountered during search will initially improve bounds for both players.
- **Maximizing Player:** When evaluating max node, if max nodes value is greater than or equal to beta, there is no need to explore further because opponent will never choose this branch because its already worse than known branch. So, it can be pruned, and search returns current node's value as result.
- **Minimizing Player:** When evaluating opponent, if current node's value is less than or equal to alpha, it implies that current player, player would never choose this branch. Branch can be pruned.
- **Update Bounds:** As search proceeds, alpha and beta are updated to reflect best values encountered so far for max and min players.

Example:



1. Start at node A with α = -∞ and β = +∞.
2. Move to node B:
   - B is a MAX node. Explore its children.
   - Evaluate node E with a value of 3. Update α = 3.
   - Move to node F:
   - Evaluate node F with a value of 5. Update α = 5.
   - Move to node G:
   - Evaluate node G with a value of 4. Update α = 5.
   - Prune node G because β = +∞ is greater than α = 5. No need to explore further.
   - Node B has two children (E and F) evaluated.
3. Return to node A:
   - Node C and D are not explored because we know they won't affect the final decision.
   - Prune nodes C and D because β = +∞ is greater than α = 5.
4. We have explored nodes E, F, and G, and pruned nodes C and D, reducing the number of static evaluations.

This example illustrates how Alpha-Beta Pruning efficiently eliminates branches that cannot possibly change the final outcome of the minimax search, thus reducing the number of leaf node evaluations.

QUESTION 11: